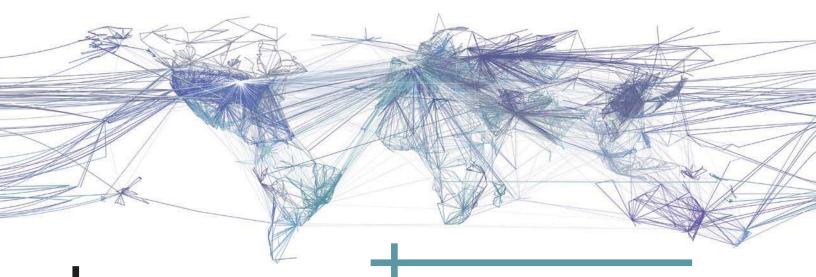


Our brain is a Big Graph, a network of trillions of neurons connected by synapses, whose topology shares common characteristics with other graphs, such as social networks. Can we unlock the secrets of our neural processing using graph theory and Big Data technologies? (Image reprinted from [1].)

Big Graphs

Paul Burkhardt



graph is a group of associated objects represented by a network of vertices and edges, where a

FIGURE 1. Graphs arise naturally from physical networks, such as the flight paths between airports. (Design: Thirst. Project: O'Hare Terminal 5 Mezzanine Mural. Client: Westfield Development. Illustration built using Processing Data by http://OpenFlights.org [2].)

vertex is an object and an edge connects a vertex to another vertex to denote their pairwise relationship. Graphs arise naturally from physical networks, such as the roads and highways connecting our cities, the power grid that transfers electricity to our homes, and the flight paths between airports (see figure 1). Biological systems also exhibit graphs, such as the interactions between proteins (see figure 2) and the conformational topology of polymers. The neurons in our brain send signals over synapses, forming one of the largest natural networks in existence. We also engineer networks from the minute electronic circuitry in microprocessors to the massive digital network of the Internet, displayed as a graph in figure 3, facilitating communication between computers all over the world.

(a)

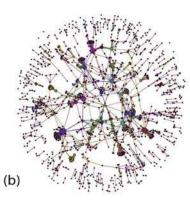


FIGURE 2. (a) Biological systems also exhibit graphs, such as the interaction between proteins. Above is a yeast protein interactome. (Graph created with Gephi, http://www.gephi.org.) **(b)** Above is a *Mycobacterium tuberculosis* interactome. (Image reprinted from [3].)

Graphs are everywhere

A graph can also be constructed from abstract and less obvious sets of relationships. For example, this article can be visualized as a graph of words. While reading this sentence, connect any pair of words co-occurring in a span of four words but counting only nouns and verbs. Our simple word graph in figure 4(a) reveals a number of cliques with a maximum size of four vertices. A clique is a group of vertices that are all pairwise connected, indicating the vertices are closely associated because each vertex is directly connected to any other. An interesting structure emerges where two of the largest cliques around the predicates connect and counting share the words vertex, thus tying any pair of vertices in this structure by two edges or less (see figure 4(b)). We can infer that connecting pairs of words in the sentence is closely associated with counting nouns and verbs, but reading is not closely associated to nouns and verbs in this context because <u>reading</u> is separated by no less than three edges to either <u>nouns</u> or <u>verbs</u>, despite the obvious grammatical relationship.

Word co-occurrence graphs are an abstract representation of written language that can help expose semantic meaning by machines. Another less obvious utilization of graphs is solving the *shortest superstring problem*—the task of creating the shortest string that contains each substring from a set of *n* substrings. If the length of the superstring did not matter, then the problem is trivially solved

by concatenating all the substrings. Constructing the shortest superstring that contains each substring exactly once is much harder but has applications in data compression and genome assembly. A brute-force method that shortens a superstring by the overlap between substrings must do so for all *n!* possible superstrings, which quickly becomes intractable (e.g., 15! is over one trillion).

The shortest superstring problem can be solved by creating a graph where

vertices are the *n* substrings and all pairs of vertices are connected by edges with a weight given by the longest suffix of one vertex that is equal to the prefix of the other and a direction in that order, then finding a Hamiltonian path that visits each vertex once while maximizing the overlap (also known as the Traveling Salesman Path Problem). But finding a Hamiltonian path is in the class of NP-complete (i.e., nondeterministic polynomial

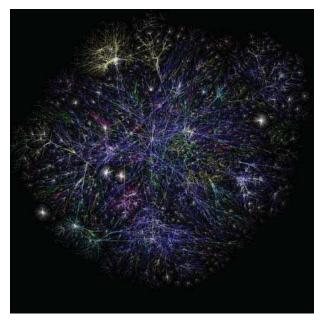


FIGURE 3. We engineer networks, such as the digital network of the Internet, displayed above as a graph. (Image from [4].)

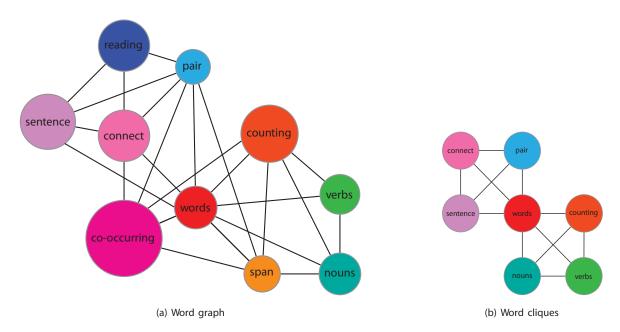


FIGURE 4. (a) Texts can be visualized as a graph of words, such as the graph above of the co-occurrence of words in a sentence from this article. (b) These word cliques (a clique is a group of vertices that are all pairwise connected) from figure 4(a) reveal associations between words. Here, they reveal that connecting pairs of words in the sentence is closely associated with counting nouns and verbs.

time-complete) problems for which efficient solutions are not known.

A special case where each substring has length *k* over an alphabet of size *n* is more tractable. This problem can be solved by constructing a de Bruijn graph where each *k*-length substring is an edge that begins from its (k-1)-length prefix and ends at its (k-1)-length suffix, then finding a Eulerian cycle—a path that traverses each edge exactly once before returning to the origin. (Eulerian cycles are inspired by Euler's 1735 solution to crossing the Seven Bridges of Königsberg over the river Pregel which started the study of graph theory.) The de Bruijn graph in figure 5 admits a Eulerian cycle, just follow the labeled edges in order and concatenate the first symbol in each edge to construct the cyclic superstring 0000110010111101, representing all sixteen k=4 length substrings for an alphabet of 0 and 1. The graph by de Bruijn is an important method used in DNA sequencing where possibly billions of k-mers (i.e., substrings of k-length) must be assembled to construct the final genome sequence.

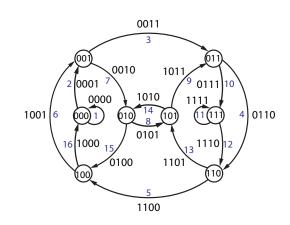


FIGURE 5. This de Bruijn graph admits a Eulerian cycle—a path that traverses each edge exactly once before returning to the origin. This type of graph can be used to solve the shortest superstring problem and is used in DNA sequencing. (Image reprinted by permission from Macmillan Publishers Ltd: Nature Biotechnology, available at http://www. nature.com/nbt/index.html, Compeau PE, Pevzner PA, Tesler G, "How to apply de Bruijn graphs to genome assembly," doi: 10.1038/nbt.2023, fig. 2, 2011 [5].)

What can graphs tell us?

A graph can be a beautifully complex and intriguing topology of interconnected pathways, alluding to hidden meaning and secrets available only to the intrepid willing to walk the edges. Often the graph resembles little more than a hair ball, such as the graph of the World Wide Web in figure 6, obfuscating insight by the seemingly infinite number of circuitous paths. But Google's search engine, for example, is based roughly on the concept of randomly following links from one web page to another in a gigantic web graph, ranking each page according to the popularity of the pages that link to it, and returning surprisingly accurate search results.

Social interactions can be symbolized by graphs (e.g., the Twitter graph in figure 7) and inspire colloquial phrases such as *small world* and *six degrees of separation*, indicating that we are all connected by just a few associates. The topology of our social networks was discovered to be more resistant to failure when nodes or links are removed, which can result in the dissociation of communities or the disruption of pathways, and to be better

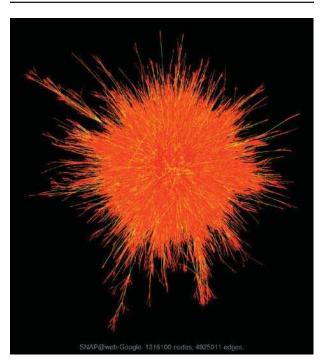


FIGURE 6. Some graphs, such as this World Wide Web graph, are so complex that they resemble little more than a hair ball. (Image from [6].)

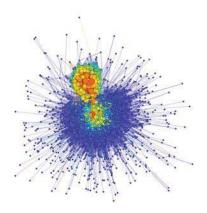


FIGURE 7. Social interactions, like those on Twitter, can be symbolized by graphs. (Graph created with Gephi, http://www.gephi.org.)

at disseminating information than other graph topologies [7, 8]. These small-world graphs have more cliques and shorter paths, but it is the severe inequities among the vertices that explain why rumors and disease quickly spread throughout these networks. Because a few vertices incur the vast majority of edges, acting as hubs, many low-degree vertices with only a few direct neighbors are able to exchange information easily [8].

Such small-world graphs can be found in many real-world networks. For example, the hub structure can be found in the network of US airports where, according to 2012 data, 80% of passengers are serviced by only 50 out of nearly 20,000 airports [9, 10]. The small-world graph properties can also be found in neural networks, such as that of the soil nematode Caenorhabditis elegans (shown in figure 8), implying these graph properties have an evolutionary benefit [7]. Thus, out of complex, unordered, and decentralized interactions, logic and purpose arise. Small-world graphs develop naturally without any centralized control or predefined order but rather from preferential attachment where popular nodes become more popular over time just as our network of roads started as decentralized clusters localized to cities and towns, eventually connecting to other clusters creating hubs around the big cities.

Graphs are truly everywhere and can be literally constructed from any data. But graphs do

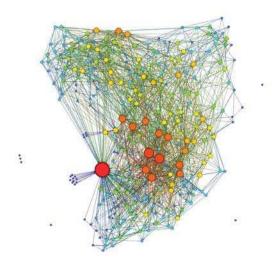


FIGURE 8. Small-world graph properties can also be found in neural networks, such as this one of the soil nematode *Caenorhabditis elegans*, implying that these graph properties have an evolutionary benefit [7]. (Graph created with Gephi, http://www.gephi.org.)

not by themselves add information; instead, they help to organize data as a collection of (key, value) pairs that effectively encode binary relationships which, when analyzed in whole, can reveal surprising insight to complex interactions. Algorithms that discover cues from navigating the graph have been studied since the days of Euler and the Seven Bridges of Königsberg (only two bridges from his day still stand) nearly three centuries ago. Searching a complex network is an exemplary application of graph algorithms and one familiar to us each time we use GPS navigation to compute the best route from one location to another. But graph algorithms are computationally challenging because of their irregular structure and combinatorial expansion.

One of the simplest data structures is a binary tree in which each node begets two more nodes. The branches of this graph expand in powers of two, so after just 16 generations, there are already 131,071 nodes, and another 16 generations later, there are over eight billion nodes. In most graphs, the branches are not regular and expand much more quickly. In many real-world graphs, the disparity in degree distribution creates significant resource contention during computation. The powerful analytic capability of graph algorithms has

motivated the design for efficient parallel processing of graphs in high-performance computing (HPC) systems. Fields such as genomics, molecular dynamics, and data science are utilizing many of these HPC graph algorithms to analyze their large and complex data sets. The rising tide of Big Data has created interest in applying graph-theoretic approaches in these fields and many others. But as data sets get larger, the challenges to graph processing increase to a point where even the most powerful HPC systems will buckle under the task of graph analysis on Big Data.

Big Graphs

The introduction to Big Data gives a sense of the massive scale of some of these data sets which would create very big graphs. On any given day the web contains about 50 billion web pages (cf. http://www.worldwidewebsize.com), and if we estimate an average of 20 URL links per page, the web graph would have one trillion edges. In 2008, Google had already claimed to have indexed a total of one trillion pages. In October of 2012, Facebook announced that their social media site had reached one billion active monthly users, connecting one out of seven people on the planet, and since 2004, there have been 140.3 billion friend connections. In early 2013, Facebook announced their Facebook Graph Search to harness the Big Data graph information collected in their social network which could include the more than one trillion "likes" made by their users.

The computational resources for searching the web or the Facebook network are hidden in secret data centers built by Google and Facebook. But in 2010, Google published their Pregel paper for processing large-scale graphs [11]. In this paper, Google described their distributed-memory approach, which follows the bulk synchronous parallel (BSP) model of computing rather than the parallel random access machine (PRAM) model traditionally favored for graph algorithms. A distributed-memory system is a cluster of machines, each with their own private memory, and data residing in the memory of one machine must be explicitly communicated to another machine. Increasing the memory of such a distributed-memory system only requires

connecting more machines. In contrast, a shared-memory system has a single pool of memory that is accessible to all machines, while each machine also has a small portion of private memory. Communicating data changes to all machines, therefore, simply requires updating the data in the pool. But a protocol must be enforced to ensure data remains consistent, especially when one machine has loaded data into its own private cache but, before it can process that data and replace the modifications in the pool, another may have already made changes. This cache-coherency protocol makes it much more difficult to scale shared-memory computers.

Another limiting factor is that a central processing unit (CPU) has a memory address limit. For example; the Intel Xeon E5 has a 46-bit address space [12]; therefore, a system comprised of these CPUs can have no more than 64 terabytes (TB) of globally-shared memory. It is not surprising that Google's Pregel favors the distributed-memory model. But the Big Graph challenge does not end here.

The problem with big brains

One of the largest physical networks is our own neural network, the human connectome, depicted in figure 9. If we count neurons as vertices and synapses as edges, there are approximately 10 trillion vertices and 100 trillion edges in the human brain graph. If each edge were stored in 16 bytes, our brain graph would occupy over one petabyte (PB)—that exceeds the practical memory capacity of any computing platform today. As described below, the largest memory capacity in a supercomputer is 1.5 PB.

The human brain graph stored in bytes would occupy over one petabyte. How large is that?

(1,024) ³ bytes	=	1 gigabyte (GB)
1,024 GB	=	1 terabyte (TB)
1,024 TB	=	1 petabyte (PB)

Leaving the memory issue aside, if we traversed edges at a pace of one every millionth of a second (microsecond) it would us take over three years to visit each neuron without ever retracing a step. This rate of one million edges per second is clearly impractical, but considering the fastest network technologies available have microsecond latency between one network interface to another, it will require careful implementation on a many-processor

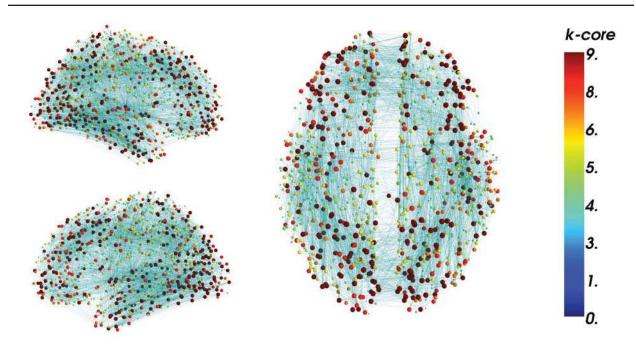


FIGURE 9. One of the largest physical networks is our own neural network, the human connectome. Copyright © 2011 Gerhard, Daducci, Lemkaddem, Meuli, Thiran, Hagmann [13].

supercomputer to overcome the latency costs incurred by traversing all the edges.

A typical CPU, the "brain" we are familiar with in our personal computers, can operate at gigahertz (GHz) frequency where the CPU can perform an instruction every nanosecond (ns) or a billionth of a second; in one microsecond, the CPU will have cycled 1,000 times. There is also a speed limitation forced upon us by physics (despite recent excitement in the now debunked "faster-than-light" neutrinos) that **light travels approximately 0.3 meters every nanosecond.** A graph with trillions of edges will necessarily be distributed across many compute racks so the distance between racks at the far ends will be a factor. This is the paradox—scaling a system to keep up with increasing data can make it more difficult to process that data!

As graphs scale with Big Data, increasing the physical memory to fit the graph may not always be practical or environmentally feasible. The Cray Titan supercomputer installed at Oak Ridge National Laboratory was the world's most powerful supercomputer in 2012 according to the Top500.org November list of that year [14]. The \$97 million Titan requires 8.2 megawatts (MW) of power [14] and over 4,300 square feet of space—an NBA basketball court is 4,700 square feet—but with a total memory capacity of 710 TB, it does not have enough memory to store the human connectome. The second most powerful supercomputer on the November 2012 list, the IBM Sequoia installed at Lawrence Livermore National Laboratory, requires 7.9 MW of power [14] and over 3,000 square feet of space. The Sequoia has 1.5 PB of memory, just enough to store the human connectome, but leaves no memory for applications that could analyze the brain graph. At a hypothetical 10 cents per kilowatt-hour, it would cost about \$7 million per year to power either of these supercomputers ($^{100}/_{MW}$ x 8,760 $^{h}/_{O}$).

Idle time on these systems is very costly, but ensuring all CPUs are performing useful work when processing a Big Graph is a daunting challenge. A single Sequoia 1.6 GHz CPU can perform 204.8 operations per nanosecond (i.e., 1.6 cycles/ns × 16 cores × 8 operations/cycle per core) [15], but if it is requesting data from another CPU that is connected 10 meters away, at least 33 ns will pass—due to the speed of light limit—before it can perform

useful work. That is a waste of 6,831 operations for just *one* CPU; there are 98,304 CPUs in Sequoia!

Graphs at Big Data scales will demand substantial system resources for processing and storing, but reality forces limitations on budget, which includes the up-front cost of an installation, lifecycle support and maintenance, and the power required to keep the lights blinking, disks whirring, and fans humming. These systems will inevitably face hardware and software failures, making fault tolerance more imperative because restarting an algorithm on a petabyte or larger graph is very costly in time and resources. We need new approaches if we are to analyze Big Graphs.

Exception! Out of memory

In addition to the limitations of power, space, and cooling, there are hardware constraints to scaling the memory capacity of a system. Data is processed by entering through the CPU pins that interface the CPU to the memory bus. The number of pins is physically limited, which results in a memory bandwidth wall. In addition, a memory controller that mediates the data between main memory and the CPU has a fixed number of memory channels for transferring data because of the electrical constraints in the circuitry. These constraints force a hard ceiling on the maximum memory capacity for a processing unit. Using the Intel Xeon E5 again as an example, it supports four channels with each channel supporting three memory slots for a total of 12 slots per CPU, and at 8 GB per slot [12], such a dual Xeon motherboard would have 192 GB of memory.

An adjacency list is a common graph data structure that uses an array for storing vertices and a doubly-linked list for storing the adjacency or neighborhood of each vertex. This adjacency list requires on order of n + 4m memory locations for n vertices and m edges, and for large graphs, each location would require 8 bytes. To store the brain graph entirely in memory using the adjacency list (using 100 trillion = 2m), a system would need over 8,000 of these Intel Xeon E5 motherboards and 204 racks to house them; there are 200 racks in the Titan supercomputer. The cost in memory alone for this system would be almost \$20 million at \$100 per

memory slot. The 96-rack Sequoia supercomputer supports a maximum of 64 GB of memory per CPU with 1,024 CPUs per rack, which will be useful in the event that we discover a life form with a 6 PB brain graph . . . but no bigger!

If the graph cannot fit in the aggregate memory, it cannot be processed. The conventional solution is to increase the system size (i.e., add more compute boards), but that will exacerbate the latency costs, making it harder to send data from CPU to CPU to keep them busy. Bottom line: It will be difficult to scale memory in this manner if data continues to increase at exponential rates.

We can store and process Big Graphs on modest computing clusters where the graph data itself resides on disk. If the graph gets too big, then more or bigger disks can be easily added since disks have much greater data capacity than memory modules and many more drives can be attached (possibly over 100 with port multipliers). But accessing data on disks can be one million times slower than accessing it in memory. Algorithms for graphs on disk must amortize the higher latency of disk access by increasing the throughput of data. These algorithms minimize the amount of random access to avoid a frenzy of mechanical movement from disk heads seeking for data sectors. To do this effectively, the algorithms organize data in large sequential blocks because disk heads can efficiently scan data in this manner. This external memory (i.e., out-of-core) processing was first developed in the 1980s to cope with the growing disparity in both cost and performance between disk and memory, so the problem of insufficient memory is not new [16]. Processing graphs too big to fit in memory appeared in the 1990s as streaming [17] and parallel disk model [18, 19] applications.

Big Graphs in the cloud

Open-source cloud technologies inspired by Google publications [20, 21] are being leveraged to solve Big Data problems in both industry and government. The Apache Accumulo project (http://accumulo.apache.org), originally an internal research project at the National Security Agency (NSA), can be used as a graph database that can scale with disk capacity while providing security, availability, and fault tolerance. A Big Graph can

be stored in Accumulo as a collection of sorted edges and queried using the Accumulo interfaces for scanning records. The Hadoop MapReduce (http://hadoop.apache.org) programming framework can be combined with Accumulo for added processing power. A straightforward approach is to filter out edges from Accumulo (i.e., extract a subgraph) which can then be analyzed by MapReduce applications.

Storing a graph as edges is natural in (key, value) repositories, like Accumulo, since an edge is a vertex pair (i.e., the end points). Tables in Accumulo are distributed as a set of tablets, often many tablets on a single host in a cluster of multiple hosts. Each table is stored on disk in the Hadoop Distributed File System (HDFS), which replicates all data across the cluster to tolerate faults. Accumulo keeps track of the location of all tablets and can rebalance the distribution on demand. The tablets can migrate from one host to another depending on the load distribution or host failures. The (key, value) records are sorted in each tablet, and tablets can be grouped dynamically so scans can efficiently access only relevant subsets of data.

In real-world graphs such as the social and neural networks discussed earlier, the degree for a few vertices can be much larger than the rest, resulting in skew distribution of tablets. This skew creates a *hot spot* or bottleneck since the majority of queries will access only a few of the tablets. Additionally, adjacencies would be larger for Big Graphs, increasing the time needed to scan all entries in a tablet. In Accumulo, a large adjacency can be distributed across multiple tablets to enable greater parallel processing, and the tablet sizes can be controlled for better latency and less resource contention. The locality can be set—that is, tablets can be grouped based on types of edges (i.e., scan blue versus green edges)—to skip over data that is not relevant to the query.

Updating edges in the Accumulo edge table can be accomplished using the online ingest interface or the offline bulk load operation. The latter, as the name suggests, is reserved for large, wholesale updates that are completed in bulk. The ingest interface provides a timely, low-latency mechanism which inserts updates that are globally sorted in periodic compaction operations; deleted edges are

removed after the compaction step. In the event that a tablet fails before sorting its entries, the updates can be recovered from the write-ahead logs.

The MapReduce programming model is effective for distributed problems that can be decomposed into many independent tasks. The map step processes input data into a collection of (key, value) pairs which are then sorted and combined in the reduce step. By minimizing interdependency between processing elements, the amount of communication over the network is decreased and more time can be spent on actual processing—maximizing the computation-to-communication ratio. Increasing the number of compute resources should proportionately decrease the processing time to just about the time required to communicate data between the map and reduce steps.

The canonical example of an embarrassingly parallel MapReduce algorithm that minimizes communication is the simple word-count pattern described in the seminal MapReduce article by Google [21]. The algorithm counts the occurrence of every word in a large corpus of documents where each document is split into blocks of lines and distributed to many processing elements. The blocks are processed simultaneously by many independent map tasks which output (word, 1) pairs for each word. These pairs are collected and summed in the reduce tasks to calculate the count for each word. You could run the MapReduce word-count algorithm on this article to output how many times the words "big," "data," and "graph" were used, but the effectiveness of MapReduce is better realized on very large data sets where the latency from disk access can be amortized.

Developing effective graph algorithms in the MapReduce programming model requires "thinking in MapReduce," which may seem unnatural at first. But this recasting of conventional graph algorithms into counting (key, value) pairs in MapReduce can make it possible to analyze massive graphs residing on disk [22, 23] by exploiting locality. The complexity involved in explicitly communicating and sharing data to analyze large graphs in BSP and PRAM systems is eliminated in MapReduce because the framework manages the data movement. The result of this simpler programming interface is that it can be more difficult

to express efficient algorithms in MapReduce. But combining both Accumulo and MapReduce is a practical approach for storing, extracting, and analyzing Big Graphs. Here in the Computer and Information Sciences Research Group at NSA, we used this approach to demonstrate a breadth-first search at brain scale, traversing more than 70 trillion edges on a 1 PB graph [24]. This brain-size graph was nearly 20 times larger than the memory capacity in our moderate-size cluster, yet the rate of processing at this scale was the same at the scale of just one trillion edges, which fit entirely in memory.

About the author

Paul Burkhardt is a computer science researcher in the Research Directorate at NSA. He received his PhD from the University of Illinois at Urbana-Champaign. His current research interests are primarily focused on graph algorithms and Big Data analytics.

References

- [1] van den Heuvel MP, Kahn RS, Goñi J, Sporns O. "High-cost, high-capacity backbone for global brain communication." *Proceedings of the National Academy of Sciences of the United Sates of America*. 2012. doi: 10.1073/pnas.1203593109 (figure 1(b)).
- [2] Thirst. Project: O'Hare Terminal 5 Mezzanine Mural. Client: Westfield Development. Available at: http://www.3st.com/work/terminal-5-murals. (Illustration built using Processing Data by http://OpenFlights.org.)
- [3] Vashisht R, Mondal AK, Jain A, Shah A, Vishnoi P, Priyadarshini P, Bhattacharyya K, Rohira H, Ghat AG, Passi A, et al. "Crowd sourcing a new paradigm for interactome driven drug target identification in *Mycobacterium tuberculosis.*" *PLoS ONE.* 2013;7(7):1–11. doi: 10.1371/journal.pone.0039808 (figure 2).
- [4] Lyon B. The Opte Project. Map 1 [accessed 2014 Mar]. 2005 Jan 16. Available at: http://www.opte.org/maps/.
- [5] Compeau PE, Pevzner PA, Tesler G. "How to apply de Bruijn graphs to genome assembly." *Nature Biotechnology.* 2011;29(11):987–991. doi: 10.1038/nbt.2023 (figure 2).
- [6] Hu Y. Matrix: SNAP/web-Google (bipartite graph drawing) [updated 2014 Mar 12]. Available at: http://www.cise.ufl.edu/research/sparse/matrices/SNAP/web-Google.html.

- [7] Watts DJ, Strogatz SH. "Collective dynamics of 'smallworld' networks." *Nature.* 1998;393(6684):440–442. doi: 10.1038/30918.
- [8] Doerr B, Fouz M, and Friedrich T. "Why rumors spread so quickly in social networks." *Communications of the ACM*. 2012;55(6):70–75. doi: 10.1145/2184319.2184338.
- [9] Research and Innovative Technology Administration, Bureau of Transportation Statistics, U.S. Department of Transportation. "Table 1-44: Passengers boarded at the top 50 U.S. airports (a)." *National Transportation Statistics*. 2012. Available at: http://www.rita.dot.gov/bts/sites/rita.dot.gov.bts/files/publications/national_transportation_statistics/html/table_01_44.html.
- [10] Federal Aviation Administration, U.S. Department of Transportation. *Administrator's Fact Book*. 2012 Jun. Available at: http://www.faa.gov/about/office_org?headquarters_offices/aba/admin_factbook/media/201206.pdf.
- [11] Malewiz G, Austern M, Bik AJC, Dehnert J, Horn I, Leiser N, Czajkowski G. "Pregel: A system for large-scale graph processing." In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10; 2010; Indianapolis, IN. p. 135–146. doi: 10.1145/1807167.1807184.
- [12] Intel Corporation. "Intel Xeon processor E5-1600/E5-2600/E5-4600 product families datasheet-volume 1." 2012 May. Available at: http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-e5-1600-2600-vol-1-datasheet.pdf.
- [13] Gerhard S, Daducci A, Lemkaddem A, Meuli R, Thiran J, Hagmann P. "The connectome viewer toolkit: An open source framework to manage, analyze, and visualize connectomes." *Frontiers in Neuroinformatics*. 2011;5(3). doi: 10.3389/fninf.2011.00003.
- [14] TOP500.org. November 2012. Available at: http://www.top500.org/list/2012/11.
- [15] Haring RA, Ohmacht M, Fox TW, Gschwind MK, Satterfield DL, Sugavanam K, Coteus PW, Heidelberger P, Blumrich MA, Wisniewski RW, et al. "The IBM Blue Gene/Q compute chip." *IEEE Micro.* 2012;32(2):48–60. doi: 10.1109/MM.2011.108.

- [16] Munro JI, Paterson MS. "Selection and sorting with limited storage." *Theoretical Computer Science*. 1980;12(3):315–323. doi: 10.1016/0304-3975(80)90061-4.
- [17] Henzinger MR, Raghavan P, and Rajagopalan S. "Computing on data streams." 1998. DEC Systems Research Center. Technical Report No. 1998-011.
- [18] Chiang YJ, Goodrich MT, Grove EF, Tamassia R, Vengroff DE, and Vitter JS. "External-memory graph algorithms." In: *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95; 1995; San Francisco, CA. p. 139–149.
- [19] Vitter JS, Shriver E. "Algorithms for parallel memory, I: Two-level memories." *Algorithmica*. 1994;12(2–3):110–147. doi: 10.1007/BF01185207.
- [20] Chang F, Dean J, Ghemawat S,Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, and Gruber RE. "Bigtable: A distributed storage system for structured data." In: *Proceedings of the Seventh USENIX Symposium on Operating System Design and Implementation*, OSDI '06; 2006; Seattle, WA. p. 205–218. Available at: http://static.usenix.org/event/osdi06/tech/change/chang_html/index.html.
- [21] Dean J, Ghemawat S. "MapReduce: Simplified data processing on large clusters." In: *Proceedings of the Sixth Conference on Symposium on Operating Systems Design and Implementation*, OSDI '04; 2004; San Francisco, CA. p. 137–150. Available at: http://research.google.com/archive/mapreduce.html.
- [22] Burkhardt P. "Asking hard graph questions." 2014 Feb. US National Security Agency. Technical report No. NSA-RD-2014-050001v1.
- [23] Cohen J. "Graph twiddling in a MapReduce world." *Computing in Science and Engineering.* 2009;11(4):29–41. doi: 10.1109/MCSE.2009.120.
- [24] Burkhardt P, Waring C. "An NSA big graph experiment." 2013 May. US National Security Agency. Technical Report No. NSA-RD-2013-056001v1.

The Next Wave is published to disseminate technical advancements and research activities in telecommunications and information technologies. Mentions of company names or commercial products do not imply endorsement by the US Government.

This publication is available online at http://www.nsa.gov/research/tnw/index.shtml. For more information, please contact us at TNW@tycho.ncsc.mil.

0111011101110111011101110111011101110111011101110111011101110111011101 10011100111001 1110011 1100111 100111001110010 010101010101 1101)101)1010101010101 00116 00116 0011001100116 00110011001100